CS6180: Lecture 10

Robert Constable

September 21, 2017

1 Lecture Summary

We give a few more examples of *refinement style proofs*, in particular examples that go beyond first-order logic. The lecture will stress the value of the *computational content* of these constructive proofs and discuss further the concept of evidence and the notion of *evidence semantics* [2].

We will start with a look at the evidence term for a constructive proof of the proposition

$\neg\neg(P \vee \neg P)$. The realizer produced by Nuprl is exactly this:

BY

Extract of: minimal-not-not-excluded-middle normalizes to:

lambda f.(f (inr (lambda p.(f (inl p)))))

finishing with Auto.

2 Types in Constructive Type Theory (CTT) Continued

In Lecture 7 we discussed the critical idea of *canonical elements* of a type and canonical names for types. *It is essential to say when two canonical elements are considered equal.* Some types are abstract such as types for algebraic structures such as groups, rings, and fields. In these cases we do not specify what the equality relation is, but we assume there is an equality on the underlying type of objects that form the algebraic structure. The canonical names with a precise equality provide the basis for computation on the type. The data format must support the operations that characterize the type, such as addition for natural numbers. These operations give rise to *non-canonical expressions*, such as (12 + 5) or (17 x 15) and so forth.

The rules from Lecture 9 are repeated here for easy reference with some additional comments

added. We extend the theory by adding still more types, some of them have proven to be extremely useful although they are not yet implemented in other type theories. One of the most useful is the intersection type which can be used to express the new notion of *uniform evidence*.

2.1 Universes

One of the key features of Russell's type theory as expressed in *Principia Mathematica* is the notion of a *universe hierarchy,* denoted $\mathbb{U}_i$ for i ranging over the positive natural numbers. The universe hierarchy is cumulative, thus if $i < j$, then $\mathbb{U}_i$ belongs to $\mathbb{U}_j$.

It is interesting that N.G. de Bruijn [5, 6, 7] used a universe hierarchy in his implemented classical theory of mathematics based on types and a classical version of the propositions as types principle. De Bruijn found that he never needed more then three levels of universes. The Nuprl type theory allows an unbounded number of levels.

2.2 The void type

The type $Void$ belongs to every universe and has no elements.

If we are in a context where we are assuming that some element v belongs to $Void$, e.g. the hypotheses have the assumption $v : Void$, then we can conclude that $any(v)$ belongs to any well formed type A in any universe.

We think of $Void$ as the type $False$ because there is no element of this type just as there is no evidence for $False$.

2.3 Product types

Type constructor: If A and B are types, then $A \times B$ is a type, the product of A and B. The product type is in the *largest of the universes* containing A and B.

Canonical elements: If $a \in A$ and $b \in B$, then $pair(a; b)$ is a canonical element of $A \times B$.

Equality: If $(a = a') \in A$ and $(b = b') \in B$, then $(pair(a, b) = pair(a', b')) \in (A \times B)$.

Reduction rules:

$((fst\ pair(a; b)) \ = \ a) \in A$.

$((snd\ pair(a; b)) \ = \ b) \in B$.

We have seen in lecture that this type captures the idea of *conjunction* if we interpret A and B as propositions. Then the pair $A \times B$ is the type of the evidence for $A\&B$ in that order.

2.4 Dependent products

If we have a family of types indexed by elements of a type A in $\mathbb{U}_i$, that is, for each a in type A, $B(a)$ is a type, say in $\mathbb{U}_j$, then the product $x : A \times B(x)$ is a type in the maximum of the two universes. A pair $pair(a; b)$ belongs to this type if for all $a \in A$, $B(a)$ is a type, and $b \in B(a)$.

This leads to a type that I like to call the "billion dollar data type." This name made sense at the time, in 1999, with Fed chairman Greenspan, was very worried about the Y2K problem. He thought that people would hoard cash, and others had so little confidence in the software industry that he injected a billion dollars into the economy to cope with the anticipated disruptions caused by faulty software. It turns out that there were very few software problems. The dependent product data type could have guaranteed that there were even fewer. It is easy to imagine this type if we name the components, e.g. $c : Century \to y : Year \to m : Month \to Day(c, y, m)$. To implement day correctly requires a great deal of knowledge about how we keep track of the date. Handling the change to the 21st century was not so hard.

2.5 Disjoint union

Type constructor: If A and B are types, then $A + B$ is a type.

Canonical elements: If $a \in A$ then $inl(a) \in A + B$, and if $b \in B$ then $inr(b) \in A + B$.

Reduction rule: If $d \in (A + B)$ and if $u \in A$ implies $lft(u) \in G$ and $v \in B$ implies $rt(v) \in G$, then $decide(d; u.lft(u); v.rt(v)) \in G$.

2.6 Function space

Type constructor: If A and B are types, then $A \to B$ is a type.

Canonical elements: If for any $x \in A$, $b(x) \in B$, then $\lambda(x.b(x)) \in A \to B$.

Reduction rule: If $f \in (A \to B)$ and $a \in A$, then $ap(f; a) \in B$.

The propositional meaning of $A \to B$ is that A implies B, and there is at least one computable function f that converts evidence a for A into evidence $f(a)$ for B.

2.7 Dependent function space

Type constructor: If A is a type in $\mathbb{U}_i$ and for each a in A, $B(a)$ is a type in $\mathbb{U}_j$, then $x : A \to B(x)$ is a type in the maximum of the universes.

Canonical elements: If for any $x \in A$, $b(x) \in B(x)$, then $\lambda(x.b(x)) \in x : A \to B(x)$.

Reduction rule: If $f \in (x : A \to B(x))$ and $a \in A$, then $ap(f; a) \in B(a)$.

The propositional meaning of $x : D \to B(x)$ is that there is a *computable function* f that for each element d of the domain of discourse D can *compute evidence* $f(d)$ that $B(f(d))$. This captures what we as humans do to convince ourselves that the proposition $\forall x : D.B(x)$ is constructively known.

2.8 Set types

Nuprl introduced important new types into constructive type theory. One of them is the set type denoted $\{x : A|B(x)\}$. Intuitively these are the elements of type A that have property $B(x)$, but unlike in the dependent products, the type does not keep the evidence that the property B holds.

We use this type later to define the notion of *virtual evidence* that can be used to explain classical logic in terms of a rich notion of computability.

2.9 Quotient types

Another new type introduced in Nuprl is the quotient type, denoted $A//R$. This type simply changes the equality relation on its base type A. A good example is to consider what we mean constructively by the set of integers with a designated equality, e.g. the integers modulo 3 for example. What are they in set theory for example? Why is the set theoretic definition inappropriate for computer science?

2.10 Partial types

We will also look at one of the most innovative features of the type theory implemented by Nuprl, namely the *partial types*. These types include expressions which might diverge when we attempt to evaluate them, e.g. they might run forever. We introduced these in the 1980's [4, 3]. They help in giving the semantics of programming languages. These types have interesting properties and lead us to basic results in recursive function theory. We will look at these types much later in the course.

2.11 Intersection types

Another type with interesting logical content is intersection. The binary version is $A \cap B$ which is the type of all elements common to both A and B. Having evidence in this type tells us that A and B *are known for the same reason.* This is a more refined notion of "and."

2.12 Dependent intersection

This is the dependent version of the intersection type, written as $x : A \cap B(x)$. The elements are those terms that belong to $B(x)$ for all assignments of elements of A to x.

Consider the example $A : \mathbb{U}_i \cap (A \Rightarrow A)$. We can show that *the only element* of this type is the "polymorphic identity function," namely $\lambda(x.x)$. This example suggests the notion of *uniform validity* for a logical formula. A logical formula is uniformly valid when there is exactly one element that suffices as the witnessing evidence for all instances of its polymorphic form. We will expand on this notion in subsequent lectures using examples from the article *Intuitionistic completeness of first-order logic* [1].

2.13 Intuitionistic and Minimal logic

We have discussed previously that intuitionistic logic arises from Kleene's account of first-order logic by removing the famous "dreaded rule 8," $\neg\neg P \Rightarrow P$ from the axioms. In other axiomatizations of classical first-order logic, FOL, the *Law of Excluded Middle* is used instead of rule 8. That law is what we have been discussing in lecture, the assertion $P \vee \neg P$. We can derive this from rule 8 since we have seen in some considerable detail how to prove the proposition $\neg\neg(P \vee \neg P)$ in intuitionistic first-order logic, iFOL.

Minimal logic is a sublogic of iFOL that does not allow the rule that "false implies anything," $False \Rightarrow A$. The empty type, $Void$, is our version of $False$. This makes good sense in terms of evidence semantics, there is no evidence for $False$.

Friedman [8] shows how to embed iFOL into minimal logic, mFOL. McCarty [9, 10] studies this embedding in detail and shows various consequences that suggest that iFOL cannot be complete. It is quite interesting that the intersection type provides tools for proving a useful completeness theorem for iFOL that we will discuss in this course. The key new concept in this investigation is the notion of uniform validity which we will examine in more detail in due course.

References

[1] Robert Constable and Mark Bickford. Intuitionistic Completeness of First-Order Logic. *Annals of Pure and Applied Logic*, 165(1):164–198, January 2014.

[2] Robert L. Constable. The semantics of evidence (also appeared as Assigning Meaning to Proofs). *Constructive Methods of Computing Science*, F55:63–91, 1989.

[3] Robert L. Constable and Scott F. Smith. Computational foundations of basic recursive function theory. In *Proceedings of the 3^{rd} IEEE Symposium on Logic in Computer Science*, pages 360–371, Edinburgh, UK, 1988. IEEE Computer Society Press. (Cornell TR 88-904).

[4] Robert L. Constable and Scott Fraser Smith. Partial objects in constructive type theory. In D. Gries, editor, *Proceedings of the 2^{nd} IEEE Symposium on Logic in Computer Science*, pages 183–193. IEEE Computer Society Press, June 1987.

[5] N. G. de Bruijn. The mathematical language Automath: its usage and some of its extensions. In J. P. Seldin and J. R. Hindley, editors, *Symposium on Automatic Demonstration*, volume 125 of *Lecture Notes in Mathematics*, pages 29–61. Springer-Verlag, 1970.

[6] N. G. de Bruijn. A survey of the project Automath. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus, and Formalism*, pages 589–606. Academic Press, 1980.

[7] N. G. de Bruijn. The mathematical vernacular, a language for mathematics with typed sets. In R. P. Nederpelt, J. H. Geuvers, and R. C. De Vrijer, editors, *Selected Papers on Automath*, volume 133 of *Studies in Logic and the Foundations of Mathematics*, pages 865–935. Elsevier, Amsterdam, 1994.

[8] Harvey Friedman. Classically and intuitionistically provably recursive functions. In D. S. Scott and G. H. Muller, editors, *Higher Set Theory*, volume 699 of *Lecture Notes in Mathematics*, pages 21–28. Springer-Verlag, 1978.

[9] David McCarty. Undecidability and intuitionistic completeness. *J. Philos Logic*, 25:559–565, 1996.

[10] David McCarty. Completeness and incompleteness for intuitionistic logic. *Journal of Symbolic Logic*, 73(4):1315–1327, 2008.